

10408 U.S. PRO  
05/14/98

09078933 051498

Please type a plus sign (+) inside this box → ☒

PTO/SB/05 (1/98)  
Approved for use through 09/30/2000. OMB 0651-0032  
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE  
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

<b>UTILITY PATENT APPLICATION TRANSMITTAL</b>  <small>(Only for new nonprovisional applications under 37 CFR 1.53(b))</small>	Attorney Docket No. <b>AT9-98-071</b>
	First Inventor or Application Identifier <b>Geoffrey S. Lohrey</b>
	Title <b>Method and Apparatus for Fast Entry</b>
	Express Mail Label No. <b>EF 90318009345</b>

<b>APPLICATION ELEMENTS</b> <small>See MPEP chapter 600 concerning utility patent application contents.</small>	<b>ADDRESS TO:</b> Assistant Commissioner for Patents Box Patent Application Washington, DC 20231
1. <input checked="" type="checkbox"/> Fee Transmittal Form (e.g., PTO/SB/17) <small>(Submit an original, and a duplicate for fee processing)</small> 2. <input checked="" type="checkbox"/> Specification [Total Pages <b>27</b> ] <small>(preferred arrangement set forth below)</small> - Descriptive title of the invention - Cross References to Related Applications - Statement Regarding Fed sponsored R & D - Reference to Microfiche Appendix - Background of the invention - Brief Summary of the invention - Brief Description of the Drawings (if filed) - Detailed Description - Claim(s) - Abstract of the Disclosure 3. <input checked="" type="checkbox"/> Drawing(s) (35 U.S.C. 113) [Total Sheets <b>12</b> ] 4. Oath or Declaration [Total Pages <b>2</b> ] a. <input checked="" type="checkbox"/> Newly executed (original or copy) b. <input type="checkbox"/> Copy from a prior application (37 C.F.R. § 1.63(d)) <small>(for continuation/divisional with Box 17 completed)</small> <small>[Note Box 5 below]</small> i. <input type="checkbox"/> <b>DELETION OF INVENTOR(S)</b> Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. §§ 1.63(d)(2) and 1.33(b). 5. <input type="checkbox"/> Incorporation By Reference (useable if Box 4b is checked) The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered to be part of the disclosure of the accompanying application and is hereby incorporated by reference therein.	6. <input type="checkbox"/> Microfiche Computer Program (Appendix) 7. Nucleotide and/or Amino Acid Sequence Submission (if applicable, all necessary) a. <input type="checkbox"/> Computer Readable Copy b. <input type="checkbox"/> Paper Copy (identical to computer copy) c. <input type="checkbox"/> Statement verifying identity of above copies
<b>ACCOMPANYING APPLICATION PARTS</b>	
8. <input checked="" type="checkbox"/> Assignment Papers (cover sheet & document(s)) 9. <input type="checkbox"/> 37 C.F.R. §3.73(b) Statement (when there is an assignee) <input type="checkbox"/> Power of Attorney 10. <input type="checkbox"/> English Translation Document (if applicable) 11. <input type="checkbox"/> Information Disclosure Statement (IDS)/PTO-1449 <input type="checkbox"/> Copies of IDS Citations 12. <input type="checkbox"/> Preliminary Amendment 13. <input checked="" type="checkbox"/> Return Receipt Postcard (MPEP 503) <small>(Should be specifically itemized)</small> * Small Entity Statement(s) <input type="checkbox"/> Statement filed in prior application, Status still proper and desired (PTO/SB/09-12) 14. <input type="checkbox"/> Certified Copy of Priority Document(s) (if foreign priority is claimed) 15. <input type="checkbox"/> Other: _____ 16. <input type="checkbox"/> Other: _____ <small>* A new statement is required to be entitled to pay small entity fees, except where one has been filed in a prior application and is being relied upon.</small>	
17. If a <b>CONTINUING APPLICATION</b> , check appropriate box, and supply the requisite information below and in a preliminary amendment: <input type="checkbox"/> Continuation <input type="checkbox"/> Divisional <input type="checkbox"/> Continuation-in-part (CIP) of prior application No: _____ Prior application information: Examiner _____ Group / Art Unit: _____	

<b>18. CORRESPONDENCE ADDRESS</b>			
<input type="checkbox"/> Customer Number or Bar Code Label <small>(Insert Customer No. or Attach bar code label here)</small>		or <input checked="" type="checkbox"/> Correspondence address below	
Name	Duke Yee		
Address	P.O. Box 802334		
City	Dallas	State	TX
Country	USA	Zip Code	75380
Telephone	972-997-7290	Fax	972-997-7291

Name (Print/Type)	Jeffrey S. Lohrey	Registration No. (Attorney/Agent)	31,633
Signature	Jeffrey S. Lohrey	Date	5/13/98

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

ASSISTANT COMMISSIONER OF PATENTS  
Washington, D.C. 20231

DOCKET NUMBER: AT9-98-071  
DATE: 5/14/98

Sir:

Transmitted herewith for filing is the Patent Application of:

Geoffrey O. Blandy

FOR: METHOD AND APPARATUS FOR JUST IN TIME COMPILATION OF INSTRUCTIONS

The filing fee has been calculated as shown below:

For	Number	Number	Rate	Fee
	Filed	Extra		
Basic Fee				\$790
Total Claims	32	- 20 - 12 -	x 22 =	\$264
Indep. Claims	6	- 3 - 3 -	x 82 =	\$246
MULTIPLE DEPENDENT CLAIM PRESENTED			x 270 =	\$-0-
TOTAL				\$1300

☒ Please charge my Deposit Account No. 09-0447 in the amount of \$ 1300. A duplicate copy of this sheet is enclosed.

☒ The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account 09-0447. A duplicate copy of this sheet is enclosed.

☒ Any additional filing fees required under 37 CFR §1.16.

☒ Any patent application processing fees under 37 CFR §1.17.

Respectfully submitted,

By Jeffrey S. LaBaw  
Jeffrey S. LaBaw  
Registration No. 31,633  
Intellectual Property Law Dept.  
IBM Corporation  
11400 Burnet Road - 4054  
Austin, Texas 78758  
Telephone (512) 823-0494

090793-0449

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Geoffrey Blandy

Docket No: AT9-98-071

Group No:

Serial No.

Examiner:

For: METHOD AND APPARATUS FOR JUST IN TIME COMPILATION OF INSTRUCTIONS

Assistant Commissioner of Patents

Washington, D. C. 20231

EXPRESS MAIL CERTIFICATE

"Express Mail" Label Number: EF 903 1800 93US

Date of Deposit: 5/14/98

I hereby certify that the following attached paper or fee

Specification, Declaration and Assignment

Form PTO/SB05

Transmittal Form

Recordation Cover sheet

Informal Drawings (12 sheets)

Post Card

is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to, BOX Patent Application, Commissioner of Patents and Trademarks, Washington, D. C. 20231.

Martha A. Acosta

(Typed or printed name of person mailing paper or fee)

Martha A. Acosta

(Signature of person mailing paper or fee)

09078933-051498

Docket No. AT9-98-071

## **METHOD AND APPARATUS FOR JUST IN TIME COMPILATION OF INSTRUCTIONS**

### **BACKGROUND OF THE INVENTION**

#### **1. Technical Field:**

5           The present invention relates generally to an improved data processing system and in particular to a method and apparatus for compiling instructions in a data processing system.       Still more particularly, the present invention relates to a method and apparatus for just in time compilation of instructions in a data processing system.

10

#### **2. Description of Related Art:**

          In a procedural environment, the developer often writes an application by making a series of calls to library routines provided by the system (as well as to routines written by the developer). Conceptually, the developer's code sits on  
15 top of the system code. The developer's code can access all of the system's services, but the system need know nothing about the developer's code as they are insulated from one another by one or more API layers. The developer is responsible for providing the overall behavior and flow of control of the application, with the system providing at least some of the functionality.

09078933-051498

Docket No. AT9-98-071

The procedural approach (and its more disciplined offspring, structured programming) has produced major improvements in the quality of software over the last twenty years. However, its limitations are painfully apparent in the difficulty in extending and specializing functionality. Since procedural systems do  
5 not provide flexible interfaces, developers cannot selectively change or extend the structure or behavior.

Even when extensions and modifications are made in a procedural application, it is hard to ensure that the changes will interoperate correctly with other systems that depend on the modifications. A typical result is that the  
10 solutions from one developer might not have anything in common with any other developer's solutions. Instead of a small team of experts solving a particular problem once, there are numerous teams repeatedly and unsatisfactorily (at least insofar as interoperability) addressing the same problem.

With the minimal reuse of code in a procedural system, maintenance  
15 requirements increase due to the greater amount of coding involved and the subsequent increased potential for introducing new bugs. This lack of extensibility, factorability, interoperability, and maintainability adds up to lower code and design reuse. The result is that developer productivity is severely hindered since more time and resources are spent writing code instead of solving  
20 new problems.

Docket No. AT9-98-071

On the other hand, object-oriented environments solve many of the problems associated with procedural programming. Unlike procedural programming environments, which emphasize algorithms and procedures, object oriented environments emphasize the binding of data structures with the methods to operate on the data. The idea is to design object classes that correspond to the essential features of a problem. Object oriented programming includes inheritance, which allows derivation new classes from existing ones to provide special purpose extensions and modifications from a set common objects available to all developers.

Object oriented environments include, for example, are typically polymorphic in nature, which provides flexibility to create multiple definitions for functions. This feature allows classes to be more general and hence more reusable. Polymorphism also allows new components and functions to be added easily and without disturbing the existing system. Implementing software in object oriented environments makes it possible to design software that is more extensible, reusable, and maintainable.

Docket No. AT9-98-071

Java is an object oriented programming language and environment that is designed to solve a number of problems in modern programming practice. Java is able to support applications for many types of data processing systems, which may contain a variety of central processing units and operating systems architectures. To  
5 enable a Java application to execute on different types of data processing systems, a compiler typically generates an architecture-neutral file format the compiled code is executable on many processors, given the presence of the Java run time system. The Java compiler generates bytecode instructions that are non-specific to a particular computer architecture. A bytecode is a machine independent code generated by the  
10 Java compiler and executed by a Java interpreter. A Java interpreter is a module that alternatively decodes and executes a bytecode or bytecodes. These bytecode instructions are designed to be easy to interpret on any machine and easily translated on the fly into native machine code.

Just in time (JIT) compilation involves taking bytecodes from the machine  
15 independent platform and converting them into machine code that runs on the current platform. Just in time compilation of Java bytecodes can provide impressive gains over interpretation in terms of performance. However, invariably, these gains come at the cost of increased storage utilization. Not only is the generated machine code larger than the original bytecode method, but typically the  
20 bytecode method is retained for reasons such as debug. Minimizing storage utilization is advantageous for any data processing system and is paramount for data processing systems, such as network computers, which do not employ virtual storage.

20070303 054430

Docket No. AT9-98-071

Presently available JIT compilers operate during class load or method invocation and compile entire methods. This includes all of the paths within each method whether or not they are used. Additionally, the output, compiled code of presently available JIT compilers are unable to smoothly transition between

5 methods that are compiled through the just in time compiler or jitted and those that are interpreted. This inability to provide smooth transitions leads to jitting more methods than are needed for optimal performance.

Therefore, it would be advantageous to have an improved method and apparatus for JIT compilation of Java bytecodes.

10

090718933-051493  
064750-EE82060

Docket No. AT9-98-071

### SUMMARY OF THE INVENTION

5           One object of the present invention relates generally to an improved data processing system.

          Another object of the present invention relates in particular to a method and apparatus for compiling instructions in a data processing system.

          Yet another object of the present invention relates to a method and  
10       apparatus for just in time compilation of instructions in a data processing system.

          The present invention provides a process in a data processing system for  
executing a method having a plurality of paths. The data processing system executes  
native machine code. A path is identified within the method that is being executed,  
wherein a plurality of bytecodes are associated with the path. Bytecodes are compiled  
15       for the path being executed, wherein the bytecodes are compiled into native machine  
code for the data processing system, wherein bytecodes for unexecuted paths remain  
uncompiled.

          Under the present invention, methods may be partially compiled, leaving  
infrequently executed paths within the methods in bytecode form, reducing storage  
20       use. Compilation of bytecodes as a method is being interpreted resulting in just in  
time compilation of paths that are actually executed. Bytecodes for a path that is  
compiled may be used during the same invocation of the method containing the  
path during which compilation of the bytecodes are being performed.

0007893-05498  
954750-EE682060

Docket No. AT9-98-071

### BRIEF DESCRIPTION OF THE DRAWINGS

5        The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

10

**Figure 1** is a block diagram of a data processing system in which the present invention may be implemented is illustrated;

15

**Figure 2** is a block diagram of components used in compiling Java bytecodes as they are interpreted in accordance with a preferred embodiment of the present invention;

**Figure 3** is a method block in accordance with a preferred embodiment of the present invention;

**Figure 4** is a diagram of a JIT station in accordance with a preferred embodiment of the present invention;

20

**Figure 5** is a high level flowchart of a process for processing bytecodes in accordance with a preferred embodiment of the present invention;

**Figure 6** is a flowchart of a process for determining whether to compile a method in accordance with a preferred embodiment of the present invention;

25

**Figure 7** is a flowchart of a process for processing an invoked method in accordance with a preferred embodiment of the present invention;

**Figure 8** is a flowchart of a first pass in processing a method in accordance with a preferred embodiment of the present invention;

Docket No. AT9-98-071

**Figure 9** is a flowchart of a process for performing a second pass in processing a method in accordance with a preferred embodiment of the present invention;

**Figure 10** is a flowchart of a process for compiling bytecodes in  
5 accordance with a preferred embodiment of the present invention;

**Figure 11** is a flowchart of a return process in accordance with a preferred embodiment of the present invention;

**Figure 12** is a flowchart of a time out process in accordance with a preferred embodiment of the present invention; and

10 **Figure 13** is a flowchart of a process for complex flow processing in accordance with a preferred embodiment of the present invention.

20070303 09:49:50 "EE682060"

Docket No. AT9-98-071

## DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

5           With reference now to the figures, and in particular with reference to **Figure 1**, a block diagram of a data processing system **100** in which the present invention may be implemented is illustrated. Data processing system **100** employs a peripheral component interconnect (PCI) local bus architecture. Although the depicted example employs a PCI bus, other bus architectures such as Micro Channel and ISA may be  
 10       used. Processor **102** and main memory **104** are connected to PCI local bus **106** through PCI bridge **108**. PCI bridge **108** also may include an integrated memory controller and cache memory for processor **102**. Additional connections to PCI local bus **106** may be made through direct component interconnection or through add-in boards. In the depicted example, local area network (LAN) adapter **110**, SCSI host  
 15       bus adapter **112**, and expansion bus interface **114** are connected to PCI local bus **106** by direct component connection. In contrast, audio adapter **116**, graphics adapter **118**, and audio/video adapter (A/V) **119** are connected to PCI local bus **106** by add-in boards inserted into expansion slots. Expansion bus interface **114** provides a connection for a keyboard and mouse adapter **120**, modem **122**, and additional  
 20       memory **124**. SCSI host bus adapter **112** provides a connection for hard disk drive **126**, tape drive **128**, and CD-ROM **130** in the depicted example. Typical PCI local bus implementations will support three or four PCI expansion slots or add-in connectors. Those of ordinary skill in the art will appreciate that the hardware in  
 25       **Figure 1** may vary. For example, other peripheral devices, such as optical disk drives and the like may be used in addition to or in place of the hardware depicted in **Figure 1**. The depicted example is not meant to imply architectural limitations with respect to the present invention.

0507893 054498  
 984750 EE697060

Docket No. AT9-98-071

In the depicted example, the processes described below may be executed in data processing system **100** in which a Java virtual machine is executing. In addition to the embodiment described below, the processes of the present invention are not limited to processing Java bytecodes and may be applied to real time translation of instructions from one environment to another environment.

Turning now to **Figure 2**, a block diagram of components used in compiling Java bytecodes as they are interpreted is depicted in accordance with a preferred embodiment of the present invention. Method **200** in **Figure 2** is a method containing bytecodes that are being executed. Interpreter/compiler **204** contains the processes and data structures used to interpret and compile the bytecodes within method **200** into machine code or platform specific instructions (PSI) **206** executable by the data processing system. According to the present invention, only paths executed within method **200** are jitted by interpreter/compiler **204**. Paths that are unexecuted within method **200** remain in bytecode form. PSI **206** is stored in the order of execution, which provides for improved cache efficiency.

05078933-054498  
864750-EE682060

Docket No. AT9-98-071

With reference now to **Figures 3A-3B**, a method block and a method is depicted in accordance with a preferred embodiment of the present invention. Method block **300** and method **301** is located in a Java virtual machine. Method block **300** includes PSI address **302**, lock **303**, invoker **304**, and JIT station **306**. Method **301** includes data and processes or routines. PSI address **302** points to the starting address of the compiled codes for the method identified by method block **300**. Lock field **303** is employed to lock method **301** as described below. Invoker **304** is a pointer to a routine, which performs the required set up for execution of the method and then causes the execution of the method by branching to the compiled code or by passing control to the interpreter for noncompiled code. Under the present invention, a Java method may be in one of four states: an interpret state, a begin JIT state, a continued JIT state, and a completed JIT state. In the interpret state, invoker **304** is a pointer to a routine, which performs set up work prior to passing control to the Java interpreter. In the begin JIT state, invoker **304** is a pointer to a routine that sets up a JIT station and then passes control to the interpreter/compiler. A JIT station is described in more detail below in reference to **Figure 4**. In the completed JIT state, invoker **304** is a pointer to a routine that performs set up and passes control to PSI address **302**. The requisite set up involves establishing an appropriate execution environment, which includes invoking and initializing registers in the depicted example. In the continued JIT state, invoker **304** is a pointer to routine that attempts to lock the method and, if successful, passes control the completed JIT state invoker. If the method lock is not available, the process then is directed towards the interpreter. Method block **300** includes a pointer to the bytecodes that are associated with the method. JIT station **306** contains the information needed to compile and interpret bytecodes. JIT station **306** also includes the information needed to continue compilation of unexecuted paths when these paths are executed.

Docket No. AT9-98-071

Turning now to **Figure 4**, a diagram of a JIT station is depicted in accordance with a preferred embodiment of the present invention. One or more JIT stations may be employed under the present invention. In the depicted example, JIT station **400** includes owner field **402**, and time stamp field **404**. Owner field **402** contains the address of the method currently being jitted. A zero is present in owner field **402** if JIT station **400** is available. JIT station **400** is unavailable when it is being used to actively compile or jit bytecodes. Owner field **402** prevents JIT station **400** from being used by another method when it is in a locked state. Time stamp field **404** is used to determine whether time is up while using the JIT station.

JIT station **400** also includes a JIT table **408**, which contains an entry to each instance of a bytecode for a method. In the depicted example, each entry is one byte long, and a bytecode may have more than one byte. A bytecode instruction may consist of a one byte opcode optionally followed by one or more parameter bytes, which results in more than one entry in JIT table **408**. Each entry in JIT table **408** indicates whether code joins. If code joins together with other code, a flag is turned on to indicate a join operation, which indicates a requirement for reconciliation of registers used during the compilation process. Each of these entries also indicates whether the code can be visited. In the depicted example, a 1 indicates that only one path exist to the bytecode, a 2 indicates that more than one path to the bytecode, but the bytecode has not yet been compiled. A number greater than two indicates that more than one path exists to the bytecode and that the bytecode has been compiled. Furthermore, the number is an index to the join table.

PSI buffer **410** contains the output, in the form of machine code instructions, from jitting. The instructions may be stored within PSI buffer **410** temporarily until they are stored elsewhere within the data processing system, such as another buffer. Compiler state information **412** includes information, such as the state of the Java stack and the Java variables.

Docket No. AT9-98-071

Next, join table **414** includes entries that contain PSI address field **416** and compiler state information field **418**. PSI address field **416** contains the address of the compiled code. Compiler state information field **418** contains data to reconcile registers used during the compilation process. Bugout table **420** includes entries that

5 contain PSI addresses of conditional branch instructions not yet taken in field **422**, target bytecode address field **424**, and compilation state information field **426**. Each branch in a method that is encountered, but not taken is stored as an entry in bugout table **420**. PSI address of branch field **422** contains the address of the branch not taken, while target bytecode address field **424** contains the address of the bytecode,

10 which is the target of the branch not taken. Compilation state information field **426** includes the state of the data processing system at the time the branch was not taken. These entries in bugout table **420** contain the bugout handler information needed to continue compiling bytecodes into machine code.

Turning next to **Figure 5**, a high level flowchart of a process for

15 processing bytecodes is depicted in accordance with a preferred embodiment of the present invention. The process begins by fetching a bytecode (step **500**). A determination is made as to whether there are multiple paths to this bytecode and taking appropriate action if there are multiple paths (step **502**). Step **502** is used to determine whether a join going to the bytecode from other areas or other bytecodes

20 is present. Then, the bytecode is interpreted (step **504**) and compiled (step **506**).

Docket No. AT9-98-071

Turning next to **Figure 6**, a flowchart of a process for determining whether to compile a method is depicted in accordance with a preferred embodiment of the present invention. A determination is made as to whether the method is compiler worthy (step **600**). For instance, if it has been determined that the method has been  
 5 executed frequently and recently, the method may be identified as compiler worthy . This step basically determines whether the method should be compiled using the processes of the present invention. If the method is compiler worthy , the invoker is set equal to JITIT (step **602**). JITIT the code needed to initialize, setup for, and to carry out interpretation/compilation. JITIT initializes in preparation for  
 10 interpretation/compilation and actually performs interpretation/compilation.

With reference now to **Figure 7**, a flowchart of a process for processing an invoked method is depicted in accordance with a preferred embodiment of the present invention. The process begins by determining whether the method block is locked (step **700**). This determination is made by examining a lock field, such as lock field  
 15 **303** in method block **300** illustrated in **Figure 3A**. The method will be locked if a method is being actively compiled using the JIT station. If the method is locked, the method is processed by the interpreter (step **702**). If the method is unlocked, the process then locks the method (step **704**). The method is locked using a lock, such as lock field **303** in method block **300** shown in **Figure 3A**.

20 Next, a determination is made as to whether a JIT station is available (step **706**). Determining whether a JIT station is available is performed by searching for a JIT station with a zero in the owner field and atomically updating the zero to the address of the current method block. If a JIT station is unavailable, the method is unlocked (step **708**), and the process proceeds to step **702** as described above.  
 25 Otherwise, the method address is stored in the owner field of the JIT station (step **710**).

854T50"EE632050

Docket No. AT9-98-071

Next, the JIT station is initialized (step **712**). Initialization of the JIT station requires zeroing out the join bytes and initializing the bugout table. A first pass, pass 1, is performed, during which join table and other entries in the JIT station are filled (step **714**). Next, a second pass, pass 2, occurs. This second pass involves  
5 interpreting and compiling paths in the method that is being executed (step **716**). The process then returns (step **718**).

Turning next to **Figure 8**, a flowchart of a first pass in processing a method is depicted in accordance with a preferred embodiment of the present invention. This flowchart is a more detailed illustration of step **708** in **Figure 7**. The process begins  
10 by fetching a bytecode (step **800**). A determination is then made as to whether the bytecode is a goto instruction to a target (step **802**). If the bytecode contains a goto instruction, a determination is then made as to whether the target join byte is set equal to two (step **804**). The target join byte is the entry in the JIT table for the target bytecode. If a determination is made that the target join byte is equal to two, then the  
15 process returns to step **800**. Otherwise, one is added to the target join byte (step **806**) with the process returning to step **800**.

With reference again to step **802**, if the bytecode is not a goto instruction, a determination is then made as to whether the bytecode is a return (step **808**). If a return is present, no join bytes are then updated, and the process returns to step **800**  
20 as described above. Otherwise, a determination is made as to whether the bytecode is a simple condition (SC) branch (step **810**). If the bytecode is a SC branch, a determination is then made as to whether the target join byte of the branch is equal to two (step **812**). If the target join byte is equal to two, the process returns to step **800**. Otherwise, one is added to the target join byte (step **814**). Then, a  
25 determination is made as to whether the next join byte is equal to two (step **816**). If the next join byte is equal to two, the process returns to step **800**. Otherwise, one is added to the next join byte (step **818**).

Docket No. AT9-98-071

With reference again to step **810**, if the bytecode is not a SC branch, a determination is then made as to whether the bytecode is a table/lookup switch (step **820**). If the bytecode is a table/lookup switch, complex flow processing is performed (step **822**) with the process then proceeding to step **816**, as described above. If the  
5 bytecode is not a table/lookup switch, the process proceeds directly to step **816**.

Turning now to **Figure 9**, a flowchart of a process for complex flow processing is depicted in accordance with a preferred embodiment of the present invention. **Figure 9** is a more detailed description of step **822** in **Figure 8**. The process begins by fetching the next target (step **900**). Thereafter, a determination is  
10 made as to whether the target is a join byte (step **902**). If the target is not a join byte, one is added to the target join byte (step **904**). A determination is made as to whether the target is the last target (step **906**). If the target is the last target, the process returns to step **900**. Otherwise, the process terminates. The process proceeds directly to step **906** from step **902** if the target is a join byte.

09078933 054498  
864750 88682060

Docket No. AT9-98-071

Turning next to **Figure 10**, a flowchart of a process for performing a second pass in processing a method is depicted in accordance with a preferred embodiment of the present invention. **Figure 10** is a more detailed illustration of step **710** in **Figure 7**. The process begins by setting the native instruction pointer (NIP) equal to zero, setting bytecode pointer (BCP) equal to zero, and setting JEX equal to zero (step **1000**). JEX is defined by the pointer index to the next available join table entry. A determination is made as to whether join byte[BCP] is equal to one (step **1002**). If join byte[BCP] is equal to one, then the process goes to the routine[BCP(0)] that handles that bytecode (step **1004**). Step **1004** is a compilation step that will be described in more detail below in **Figure 11** with the process then returning to step **1002**. If the join byte[BCP] is not equal to one, the process then determines whether join byte[BCP] is equal to two (step **1006**). If join byte[BCP] is equal to two, then the process allocates a join table entry setting JT[BCP] to the index of that entry and initializes that entry by setting the PSI address to point to the next instruction pointer.

With reference again to step **1006**, if join byte[BCP] is greater than two, the join byte is an index into the join table and the join table entry at that index has the PSI address and native state required to pass control to the PSI address(step **1010**).

Turning now to **Figure 11**, a flowchart of a process for compiling bytecodes is depicted in accordance with a preferred embodiment of the present invention. **Figure 11** is a more detailed illustration of step **1004** in **Figure 10**. The process begins by determining whether the bytecode is a flow altering bytecode (step **1100**). If the bytecode is not a flow, the process then compiles the bytecode (step **1002**) and sets BCP equal to Next (step **1104**) with the process then returning to step **1002** in **Figure 10**.

Docket No. AT9-98-071

20070930 05498

If the process determines that the bytecode is a flow altering bytecode, then the process determines if the bytecode is a return (step **1106**). If the bytecode is a return, then return processing is performed (step **1108**). Otherwise, a determination is made as to whether the bytecode is a goto (step **1110**). If the bytecode is a goto, 5 the BCP is set equal to the target of the goto statement (step **1112**) with the process then returning to step **1002** in **Figure 10**. Otherwise, a conditional branch exists and BCP is set equal to the path taken (step **1114**). In the depicted example, taken is the bytecode that will be next executed as a result of a conditional branch. Thereafter, a conditional branch (CB) is generated for the condition not taken in 10 which CB is equal to bugout for the condition not taken (step **1116**). Next, a bugout entry is created in bugout table **420** in **Figure 4** (step **1118**) with the process then returning to step **1002** in **Figure 10**. The bugout entry will include the PSI address of the branch, the bytecode address, and compilation state information.

Turning now to **Figure 12**, a flowchart of a return process is depicted in 15 accordance with a preferred embodiment of the present invention. **Figure 12** is a more detailed flowchart of step **1108** in **Figure 11**. The process begins by determining whether any bugouts are still present (step **1200**). If bugouts are still present, the invoker is set to a continued JIT state. (step **1202**).

If the JIT station is available, the invoker is placed in a state to setup and pass 20 control to the JIT station to continue compiling the bytecodes. If the JIT station is unavailable, the invoker is placed in a state to pass control to the interpreter/compiler. Then, the lock on the JIT station is released (step **1204**). Then, the process returns to caller (step **1206**). With reference again to step **1200**, if no bugouts are left, the method is finalized (step **1208**) with the process then proceeding to step **1204** as 25 previously described. In finalizing, the conditional branch to the bugout handler is changed to an address to continue execution at the interpreter with the bytecode pointer set to the address of the target bytecode.

Docket No. AT9-98-071

With reference now to **Figure 13**, a flowchart of a time out process is depicted in accordance with a preferred embodiment of the present invention. The time out process is used when another method needs to use the JIT station and the current owner's allotted time for the JIT station has elapsed. The process begins by fetching  
5 the next bugout entry from bugout table **420** in **Figure 4** (step **1300**). A determination is made as to whether the end of the bugout table has been reached (step **1302**). If the end of the bugout table has been reached, the method is then finalized (step **1304**). With reference again to step **1302**, if the end of the bugout table has not been reached, bugout bytecode for the target is set equal to NIP (step  
10 **1306**). Thereafter, code is generated to enter or goto the interpreter (step **1308**). In this step, code is generated to set up BCP that is needed to go back to and restore the interpreter state in the invoker with the process then returning to step **1300**.

The present invention thus provides an improved method and apparatus for compiling and interpreting bytecodes. This advantage is provided through performing  
15 compilation as a method is being interpreted. In particular, only paths that are actually being executed are being compiled or jitted. Additionally, the present invention allows for a path that has been compiled into native machine code to be used during the same invocation during which compilation is being performed. For example, if a method has a loop that is executed numerous times, the bytecode will  
20 be compiled into native code during the first iteration. During subsequent iterations, the native machine code will be executed. Additionally, under the present invention when native machine code hits a flow control instruction, (a condition branch) that directs it to non-jitted code, control is passed to a temporary step which reestablishes the state required to continue interpretation/jitting.

Docket No. AT9-98-071

Thus, the present invention also allows for methods to be partially compiled, leaving infrequently executed paths in bytecode form, reducing storage use. The present invention also provides an advantage in that cache efficiency is improved with the laying out of code in storage in the order that it is executed.

5           It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in a form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the  
10       particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media such a floppy disc, a hard disk drive, a RAM, and CD-ROMs and transmission-type media such as digital and analog communications links.

          The description of the present invention has been presented for purposes of  
15       illustration and description, but is not limited to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. For example, although the embodiment illustrated is directed towards processing Java bytecodes, the present invention may be applied to other types of languages. For example, the processes  
20       of the present invention may be applied to real time translation from one environment to another environment. The embodiment was chosen and described in order to best explain the principles of the invention the practical application to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use  
25       contemplated.

0907893-051498  
864T50-EE68060

Docket No. AT9-98-071

**CLAIMS:**

What is claimed is:

- 1 1. A process in a data processing system executing a routine having a plurality  
2 of paths, wherein the routine has includes a plurality of first type instructions and  
3 wherein the data processing system executes second type instructions, the process  
4 comprising:  
5 identifying a path within the routine that is being executed, wherein a plurality  
6 of first type instructions are associated with the path; and  
7 translating the first type instructions for the path being executed, wherein first  
8 type instructions are translated into second type instructions for execution by the data  
9 processing system, wherein first type instructions for unexecuted paths remain  
10 untranslated.
- 1 2. The process of claim 1 further comprising:  
2 executing second type instructions for a path in response to a loop back  
3 through the path during execution of the routine.
- 1 3. The process of claim 1, wherein translated instructions for the path are  
2 executed in an order and wherein the translated instructions are stored in execution  
3 order.
- 1 4. A process in a data processing system for executing a method having a  
2 plurality of paths, wherein the data processing system executes native machine code,  
3 the process comprising:

864750" E6B2060

Docket No. AT9-98-071

4 identifying a path within the method that is being executed, wherein a plurality  
5 of bytecodes are associated with the path; and  
6 compiling bytecodes for the path being executed, wherein the bytecodes are  
7 compiled into native machine code executed by the data processing system, wherein  
8 bytecodes for unexecuted paths remain uncompiled.

1 5. The process of claim 4 further comprising:  
2 executing native machine code for a path in response to a loop back through  
3 the path during execution of the method.

1 6. The process of claim 4, wherein compiled instructions for the path are  
2 executed in an order and wherein the compiled instructions are stored in execution  
3 order.

1 7. The process claim 4, wherein a JIT station is used in compiling the method.

1 8. The process of claim 4, wherein a data structure is used during compiling  
2 of the method to store information about a path as the path is compiled.

1 9. The process of claim 8, wherein the data structure stores the native machine  
2 code.

1 10. The process of claim 9, wherein the data structure is a JIT station.

1 11. A process in a data processing system for executing a method having a  
2 plurality of paths in which each path with in the plurality of paths contains a  
3 number of bytecodes, the method comprising:

964750-EE682060

Docket No. AT9-98-071

4 identifying the method that is to be executed; and  
5 compiling the bytecodes into instructions for execution by the data  
6 processing system for each path within the plurality of paths as each path is  
7 executed.

1 12. The process of claim 11, wherein unexecuted paths within the plurality of  
2 paths remain in a bytecode form.

1 13. The process of claim 11, wherein the instructions have an execution order  
2 and further comprising:  
3 storing the instructions in the execution order.

1 14. The process of claim 11 further comprising:  
2 executing the instructions for a path within the plurality of paths in  
3 response to a loop back through the path during compilation of the method.

1 15. The process of claim 11, wherein a data structure is used during compiling  
2 of the method to store information about a path as the path is compiled.

1 16. The process of claim 15, wherein the data structure stores the instructions.

1 17. A data processing system for executing a method having a plurality of paths,  
2 wherein the data processing system executes native machine code, the data processing  
3 system comprising:

4 identification means for identifying a path within the method that is being  
5 executed, wherein a plurality of bytecodes are associated with the path; and

964750 EE582060

Docket No. AT9-98-071

6            compilation means for compiling bytecodes for the path being executed,  
7        wherein the bytecodes are compiled into native machine code, wherein bytecodes for  
8        unexecuted paths remain uncompiled.

1        18.    The data processing system of claim 17 further comprising:  
2            execution means for executing native machine code for a path in response to  
3        a loop back through the path during interpreting of the method.

1        19.    The data processing system of claim 17, wherein compiled instructions for the  
2        path are executed in an order and wherein the compiled instructions are stored in the  
3        execution order.

1        20.    The data processing system of claim 17, wherein a JIT station is used in  
2        compiling the method.

1        21.    The data processing system of claim 17, wherein a data structure is used  
2        during compiling of the method to store information about a path as the path is  
3        compiled.

1        22.    The data processing system of claim 21, wherein the data structure stores  
2        the native machine code.

1        23.    The data processing system of claim 22, wherein the data structure is a JIT  
2        station.

1        24.    A data processing system comprising:

864750 "EE687060

Docket No. AT9-98-071

2 a method having a plurality of paths in which each path within the plurality  
3 of paths contains a number of bytecodes;  
4 identification means for identifying that the method is to be executed; and  
5 compilation means for compiling the bytecodes into instructions for  
6 execution by the data processing system for each path within the plurality of paths  
7 as each path is executed.

1 25. The data processing system of claim 24, wherein unexecuted paths within  
2 the plurality of paths remain in a bytecode form.

1 26. The data processing system of claim 24, wherein the instructions have an  
2 execution order and further comprising:  
3 storing means for storing the instructions in the execution order.

1 27. The data processing system of claim 24 further comprising:  
2 execution means for executing the instructions for a path within the  
3 plurality of paths in response to a loop back through the path during compilation  
4 of the method.

1 28. The data processing system of claim 24, wherein a data structure is used  
2 during compiling of the method to store information about a path as the path is  
3 compiled.

1 29. The data processing system of claim 28, wherein the data structure stores  
2 the instructions.

864750 EE682060

Docket No. AT9-98-071

1 30. A computer program product for executing a method in a data processing  
2 system, wherein the method has a plurality of paths in which each path with in the  
3 plurality of paths contains a number of bytecodes, the computer program product  
4 comprising:

5 first instructions for identifying that the method is to be executed; and  
6 second instructions for compiling the bytecodes into compiled instructions  
7 for execution by the data processing system for each path within the plurality of  
8 paths as each path is executed.

1 31. The computer program product of claim 30, wherein the compiled  
2 instructions have an execution order and further comprising:

3 third instructions for storing the compiled instructions in the execution  
4 order.

1 32. The method of claim 30 further comprising:

2 third instructions for executing the compiled instructions for the path within  
3 the plurality of paths in response to a loop back through the path during  
4 compilation of the method.

09078933-051498  
864750-EE687060

Docket No. AT9-98-071

## **ABSTRACT OF THE DISCLOSURE**

### **METHOD AND APPARATUS FOR JUST IN TIME COMPILATION OF INSTRUCTIONS**

- 5           A method and apparatus provides a process in a data processing system for  
executing a method having a plurality of paths. The data processing system executes  
native machine code. A path is identified within the method that is being executed,  
wherein a plurality of bytecodes are associated with the path. Bytecodes are compiled  
for the path being executed, wherein the bytecodes are compiled into native machine  
10   code, wherein bytecodes for unexecuted paths remain uncompiled.

864750-EE682060

**DECLARATION AND POWER OF ATTORNEY FOR  
PATENT APPLICATION**

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

**METHOD AND APPARATUS FOR JUST IN TIME  
COMPILATION OF INSTRUCTIONS**

the specification of which (check one)

X is attached hereto.

\_\_\_\_\_ was filed on \_\_\_\_\_  
as Application Serial No. \_\_\_\_\_  
and was amended on \_\_\_\_\_  
(if applicable)

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the patentability of this application in accordance with Title 37, Code of Federal Regulations, §1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, §119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s):

Priority Claimed

\_\_\_\_\_  
(Number)                      (Country)                      (Day/Month/Year)

\_\_\_\_ Yes \_\_\_\_ No

I hereby claim the benefit under Title 35, United States Code, §120 of any United

States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, §112, I acknowledge the duty to disclose information material to the patentability of this application as defined in Title 37, Code of Federal Regulations, §1.56 which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

---

 (Application Serial #)

---

 (Filing Date)

---

 (Status)

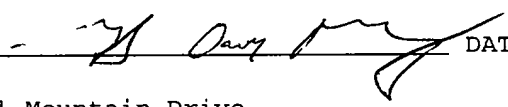
I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

POWER OF ATTORNEY: As a named inventor, I hereby appoint the following attorneys and/or agents to prosecute this application and transact all business in the Patent and Trademark Office connected therewith.

John W. Henderson, Jr., Reg. No. 26,907; Thomas E. Tyson, Reg. No. 28,543; James H. Barksdale, Jr., Reg. No. 24,091; Casimer K. Salys, Reg. No. 28,900; Robert M. Carwell, Reg. No. 28,499; Douglas H. Lefevre, Reg. No. 26,193; Jeffrey S. LaBaw, Reg. No. 31,633; Mark S. Walker, Reg. No. 30,699; David A. Mims, Jr., Reg. No. 32,708; Volel Emile, Reg. No. 39,969; Richard A. Henkler, Reg. No. 39,220; and Anthony V. England, Reg. No. 35,129; Leslie A. Van Leeuwen, Reg. No. P-42,196; Christopher A. Hughes, Reg. No. 26,914; Edward A. Pennington, Reg. No. 32,588; John E. Hoel, Reg. No. 26,279; Joseph C. Redmond, Jr., Reg. No. 18,753; Duke W. Yee, Reg. No. 34,285; David W. Carstens, Reg. No. 34, 134; and Colin P. Cahoon, Reg. No. 38,836.

Send correspondence to: Duke W. Yee, P.O. Box 802334, Dallas, Texas 75380 and direct all telephone calls to Duke W. Yee, (972) 997-7290.

FULL NAME OF SOLE OR FIRST INVENTOR: GEOFFREY OWEN BLANDY

INVENTORS SIGNATURE: 

DATE: 5/12/98

RESIDENCE: 9412 Bell Mountain Drive  
Austin, Texas 78730

CITIZENSHIP: USA

POST OFFICE ADDRESS: Same

0907893 054498  
96450 EE682060

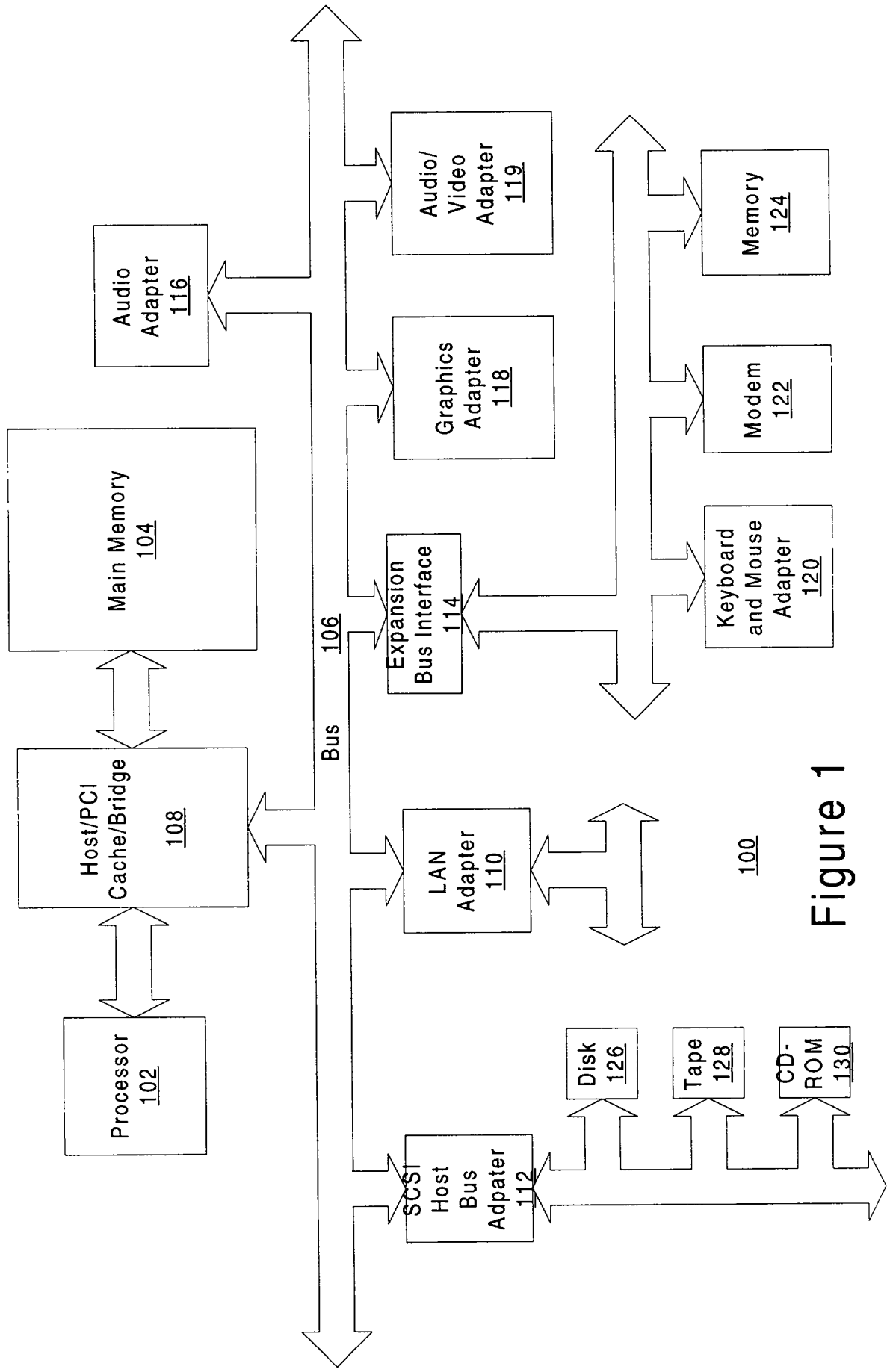


Figure 1

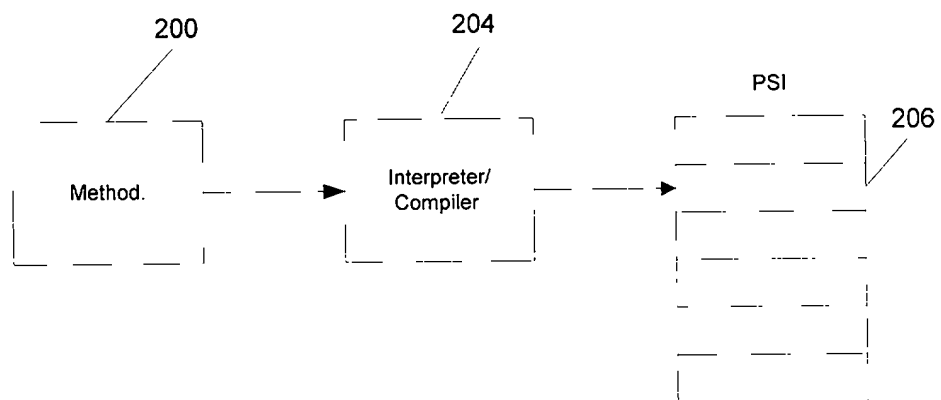


Figure 2

2009-03-06 15:00:00

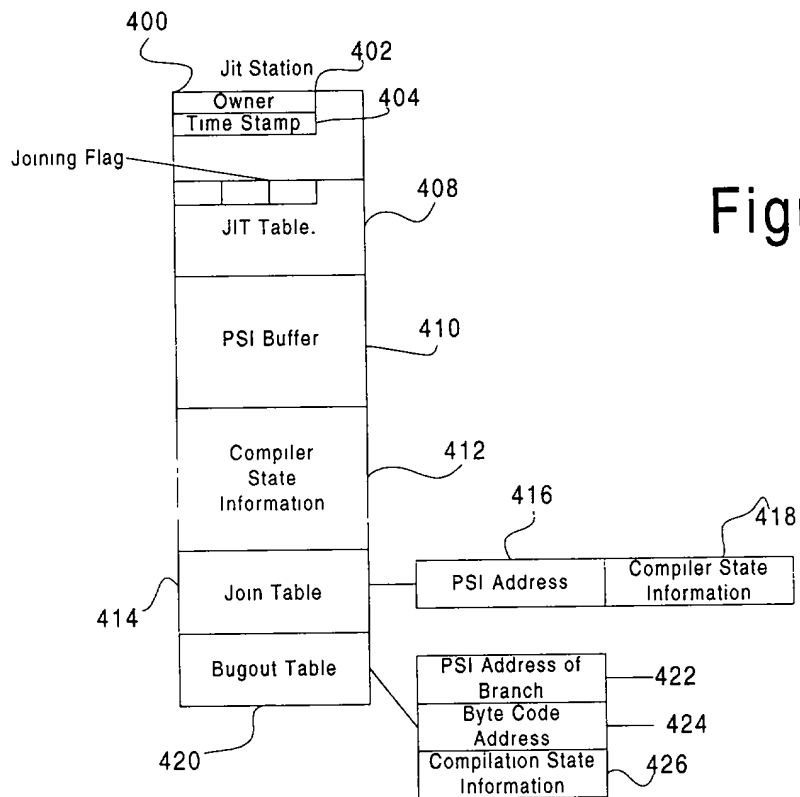


Figure 4

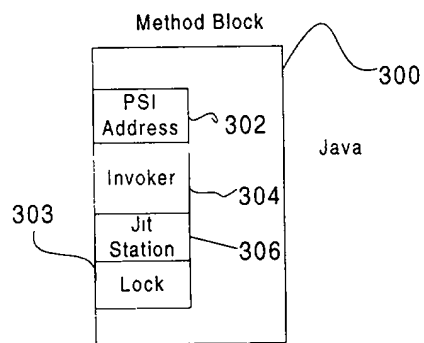


Figure 3A

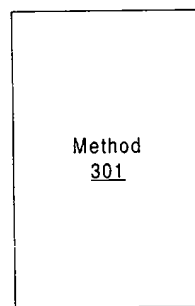


Figure 3B

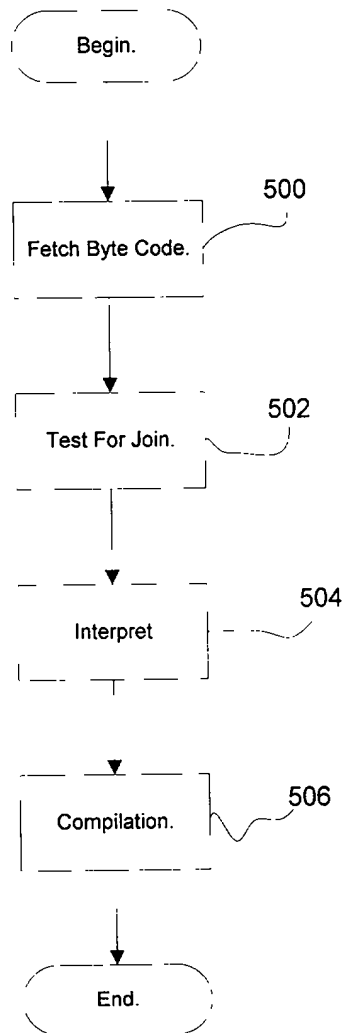
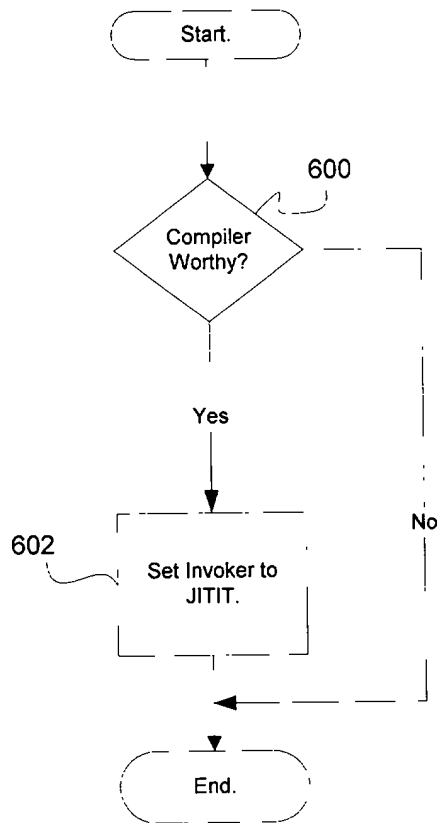
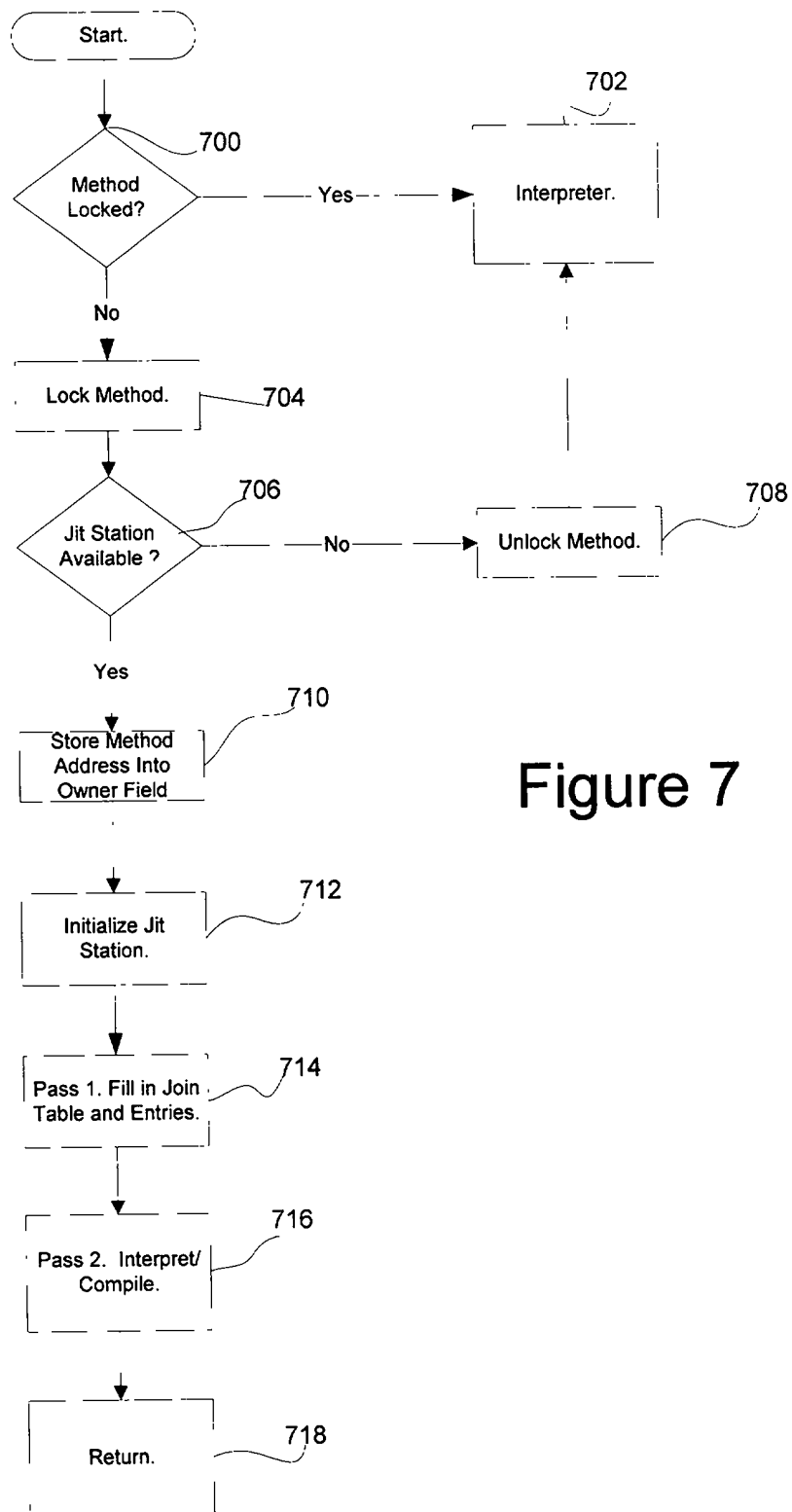


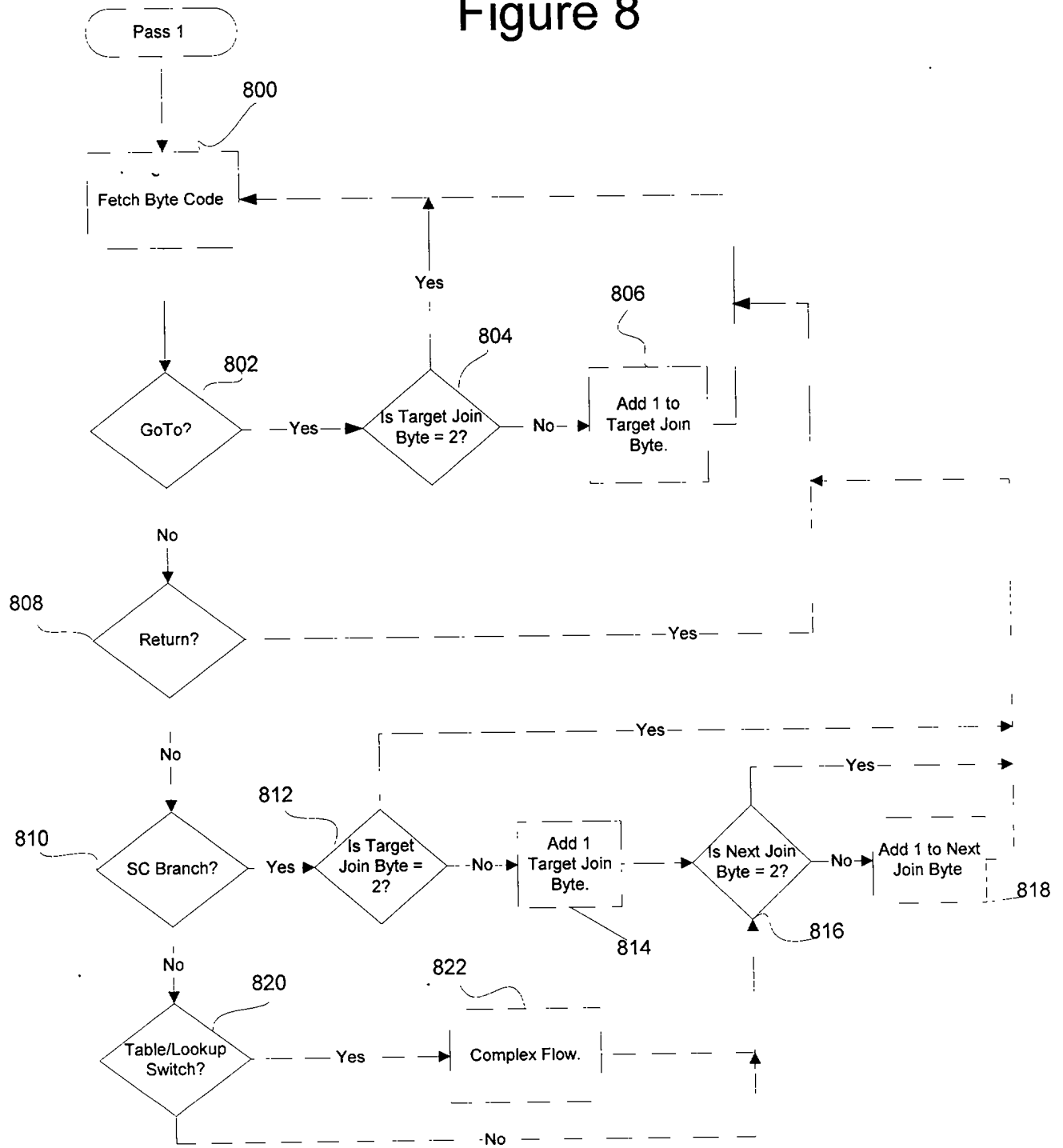
Figure 5

Figure 6





# Figure 8



0007933-051496

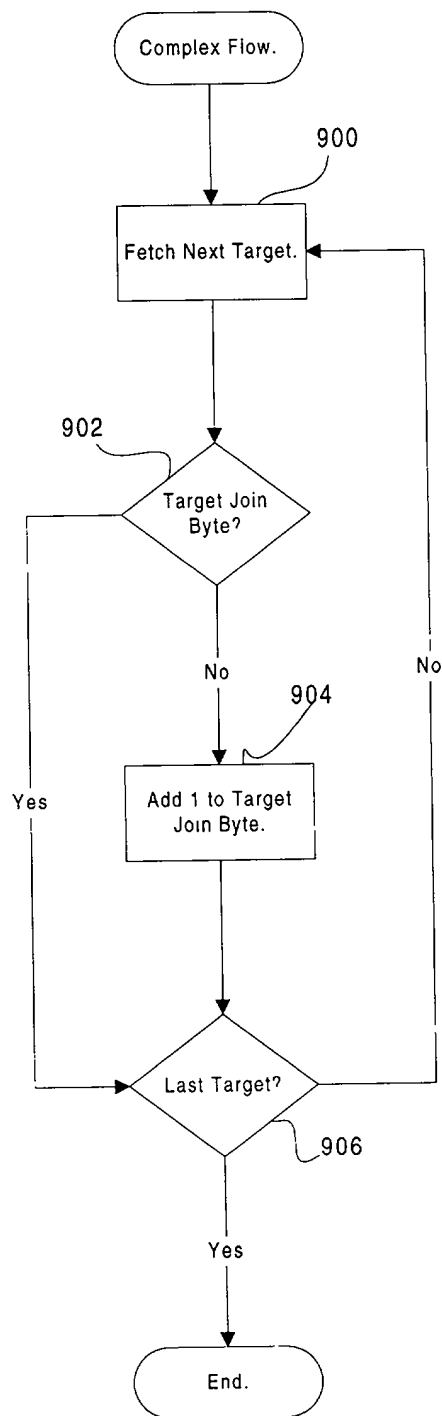


Figure 9

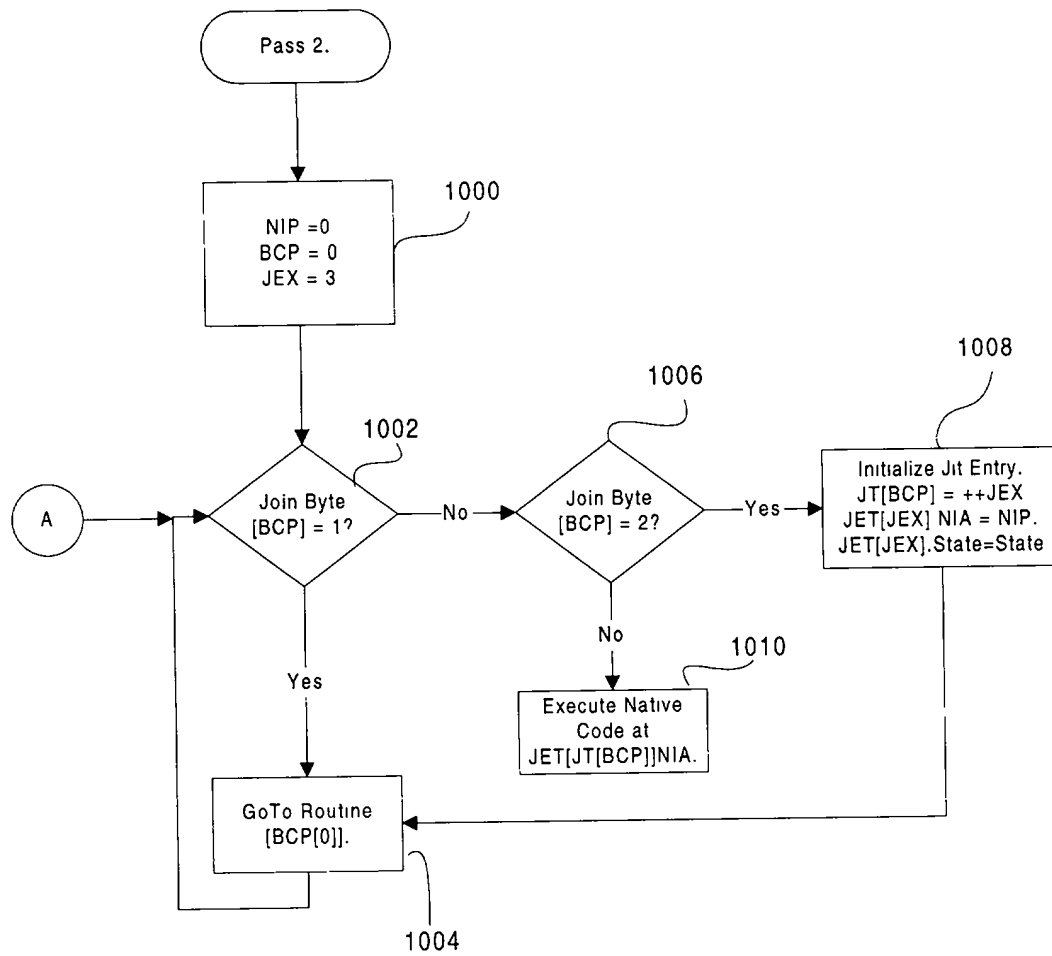


Figure 10

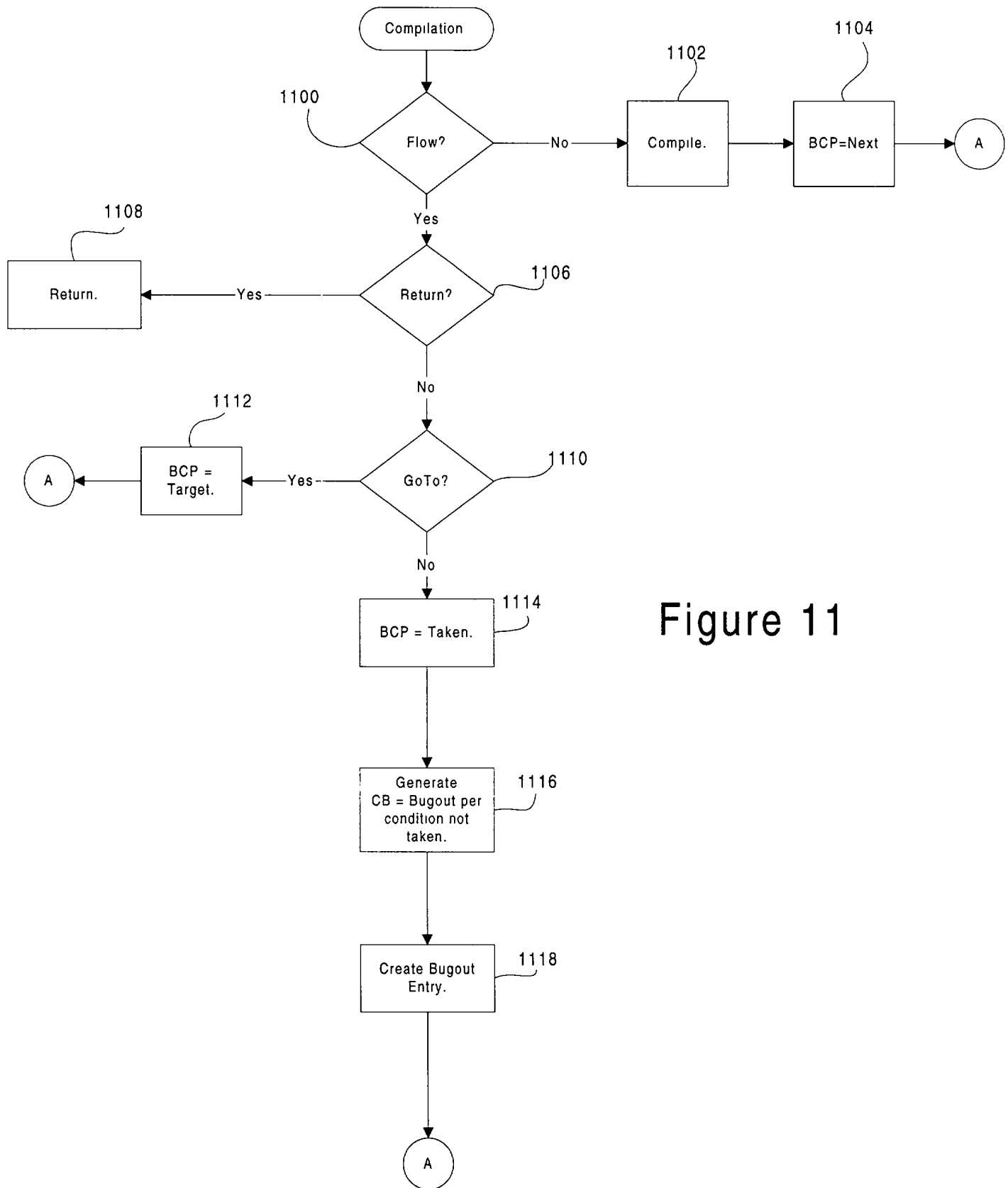


Figure 11

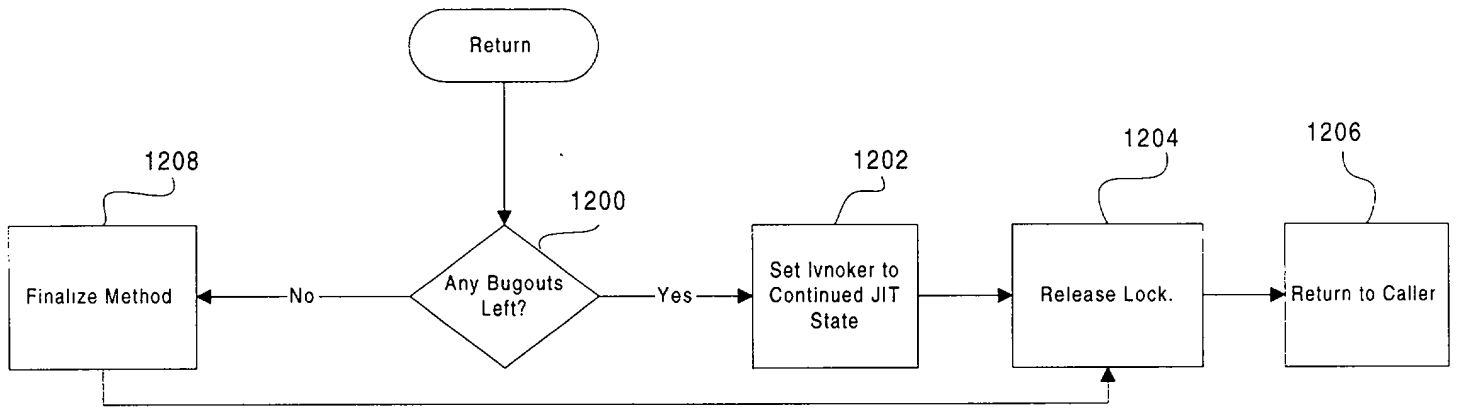


Figure 12

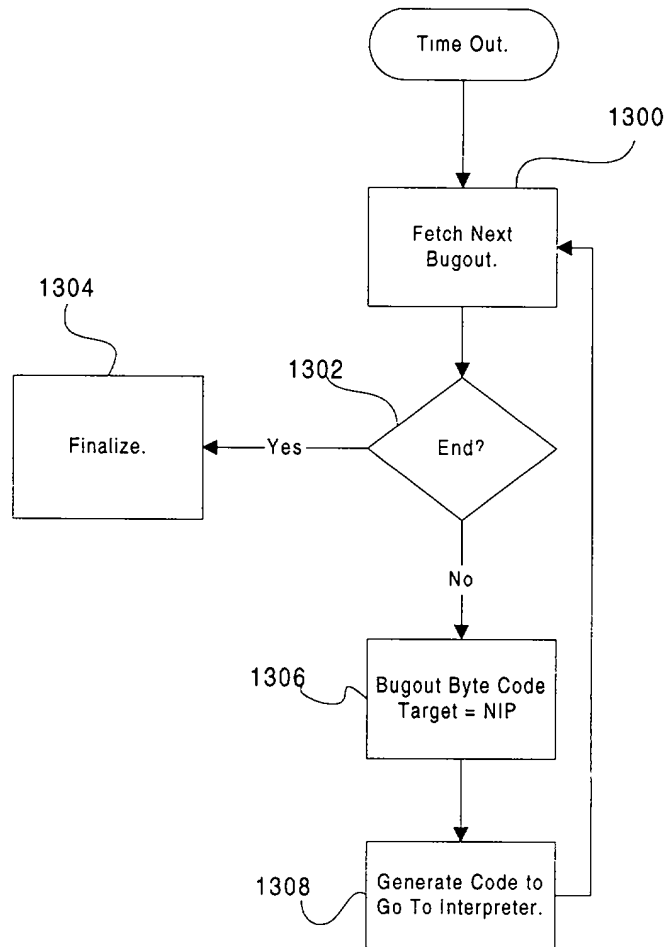


Figure 13